Formally Verified Argument Reduction with a Fused Multiply-Add

Sylvie Boldo, Marc Daumas, *Senior Member*, *IEEE*, and Ren-Cang Li

Abstract—The Cody and Waite argument reduction technique works perfectly for reasonably large arguments, but as the input grows, there are no bits left to approximate the constant with enough accuracy. Under mild assumptions, we show that the result computed with a fused multiply-add provides a fully accurate result for many possible values of the input with a constant almost accurate to the full working precision. We also present an algorithm for a fully accurate second reduction step to reach full double accuracy (all the significand bits of two numbers are accurate) even in the worst cases of argument reduction. Our work recalls the common algorithms and presents proofs of correctness. All the proofs are formally verified using the Coq automatic proof checker.

Index Terms—Argument reduction, `fma`, formal proof, Coq.

1 Introduction

Methods that compute elementary functions on a large domain rely on efficient argument reduction techniques. The idea is to reduce an argument x to u that falls into a small interval to allow efficient approximations [1], [2], [3], [4]. A commonly used argument reduction technique [1], [5], [6], [7] begins with one positive floating-point number (FPN) C_1 to approximate a number $C > 0$ (usually irrational but not necessarily so). Examples include $C = \pi/2$, π, or 2π for trigonometric functions $\sin x$ and $\cos x$ and $C = \ln 2$ for exponential function e^x.

Let x be a given argument, an FPN. The argument reduction starts by extracting χ as defined by

$$x/C_1 \quad = \quad \boxed{\chi}\boxed{\varsigma}.$$

Then, it computes a reduced argument $x - \chi C_1$. The result is exactly an FPN as it is defined by an IEEE-754 standard remainder operation. But division is a costly operation that is avoided as much as possible. Some authors (see for example [1], [3], [7], and http://www.intel.com/software/products/opensource/libraries/num.htm) introduce another FPN R that approximates $1/C$, and the argument reduction replaces the division by a multiplication so that

$$\begin{aligned} x \cdot \frac{1}{C} &\approx xR \\ &= z + s \\ &= \boxed{k2^{-N}}\boxed{s}, \end{aligned} \tag{1.1}$$

where k is an integer used to reference a table of size 2^N. This replacement is computationally efficient if

$$u = x - zC_1 \tag{1.2}$$

is an FPN [8].

Sometimes, the computed value of u is not sufficiently accurate, for example, if u is near a multiple of C, the loss of accuracy due to the approximation $C_1 \approx C$ may prevail. A better approximation to C is necessary to obtain a fully accurate reduced argument. If this is the case, we use C_2, another FPN, roughly containing the next many bits in the significand of C, so that the unevaluated $C_1 + C_2 \approx C$ is much better than C_1 alone. When (1.2) does not introduce any rounding error, the new reduced argument is not u but v computed by

$$v \approx u - zC_2. \tag{1.3}$$

To increase once again the accuracy, the error of (1.3) needs to be computed (see Section 5) to obtain v_1 and v_2 exactly satisfying

$$v_1 + v_2 = u - zC_2. \tag{1.4}$$

The last step creates a combined reduced argument stored in the unevaluated sum $v_1 + w$ with $2p$ significant bits:

$$w \approx v_2 - zC_3. \tag{1.5}$$

Whether v_1 (or v_1 and w) is accurate enough for computing the elementary function in question is subject to further error analysis on a function-by-function basis [9]. But this is out of the scope of this paper.

The Cody and Waite technique [5] is presented in Figs. 1 and 2, where $\circ(a)$ denotes the FPN obtained from rounding a in the round-to-nearest mode. Those are examples when no fused multiply-add (`fma`) is used. The sizes of the rectangles represent the precision (length of the significand) of each FPN, and their positions indicate the magnitude, except for z and C_1 whose respective layouts are only for showing the lengths of significands. The light gray represents the cancellations: the zero bits due to the fact that $|x - \circ(z \times C_1)| \ll |x|$. The dark gray represents the round-off error: the bits that may be wrong due to previous rounding(s).

Fig. 1 presents the ideal behavior. Fig. 2 presents the behavior when the significand of z is longer. Then, fewer bits are available to store the significand of C_1 if we require that zC_1 is stored exactly. The consequence is a tremendous loss of precision in the final result: as C_1 must be stored in fewer bits, the cancellation in the computation of $x - zC_1$ is smaller, and the final result may be inaccurate.

We want to take advantage of the `fma` instructions. Some machines have hardware support for it, such as machines with HP/Intel Itanium Microprocessors [1] and IBM PowerPC Microprocessors, and this instruction will also be added to the revision of the IEEE-754 standard. The last public draft can be found at http://754r.ucbtest.org/drafts/archive/2006-10-04.pdf. It is obvious that some bits of x and zC_1 will cancel each other as z is computed such that $x \approx zC_1$, but it is not clear how many of them will and under what condition(s). Consequently, if accuracy calls for $x - zC_1$ to be calculated exactly (or to more than p bits in the significand), how do we get these bits efficiently? This question is especially critical if the working precision is the highest available on the underlying computing platform.

In this paper, we will devise easily met conditions so that $x - zC_1$ can be represented exactly by an FPN, and thus, it can be computed by one instruction of the `fma` type without error. This technique is presented in Fig. 3 with the earlier drawing conventions. The cancellation is greater as C_1 can be more precise. The idea is that the rounding in zC_1 is avoided thanks to the `fma`: zC_1, a $2p$-bit FPN, is virtually computed with full

- *S. Boldo is with INRIA Saclay - Île-de-France, Parc Orsay Université, ZAC des Vignes, 4 rue Jacques Monod, Bât N, F-91893 Orsay Cedex, France, and also with LRI, Univ Paris-Sud, CNRS, F-91405 Orsay, France. E-mail: Sylvie.Boldo@inria.fr.*
- *M. Daumas is with ELIAUS, Université de Perpignan Via Domitia, 52 Avenue Paul Alduy-F-66860 Perpignan Cedex, France. E-mail: marc.daumas@ens-lyon.org.*
- *R.-C. Li is with the Department of Mathematics, University of Texas at Arlington, 445 Pickard Hall (PKH)-P.O. Box 19408, Arlington, TX 76019-0408. E-mail: rcli@uta.edu.*

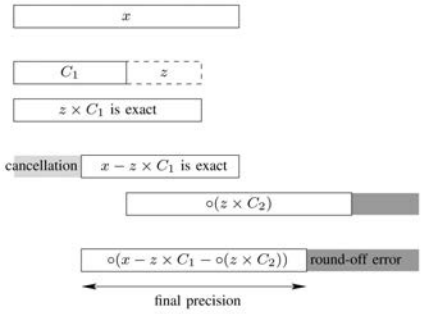

Fig. 1. The reduction technique works for a sufficiently small z.

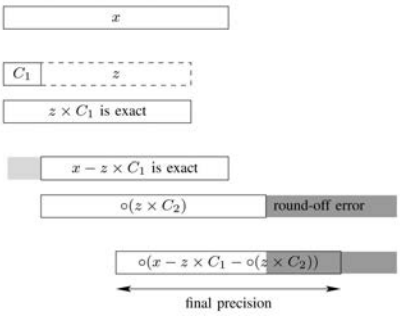

Fig. 2. The Cody and Waite technique fails as z grows, i.e., u is not accurate enough.

precision and then subtracted from x. This subtraction is proved to be exact as $x \approx zC_1$. The fact that $x - zC_1$ is an FPN is used by libraries [1], [7]. Harrison first sketched this fact [10] and later proved it formally [11]. His theorems and proofs were not published due to their length and their "messiness." There is no reason to shade some doubt on Harrison's results, and our results share many features with them, but it is difficult to precisely compare these works as Harrison's statements were never made explicit. Moreover, our work provides precise statements for all our theorems, including when Underflow conditions arise, and we provide results about the next step of the argument reduction.

The motivations of this work are similar to those presented in [8], and Section 2 recalls briefly some useful prior art in [8] and [12]. However, the rest of the paper presents new results. The theorems and their proofs are different from the ones presented in [8]. More precisely, the idea is basically the same, but the conditions on the constants are much weaker and easier to grasp. The results in Section 5 are entirely new. All results are formally proved so that no mistake could be buried in our proofs and all the Underflow conditions are explicitly stated. The simplifications of the results are due to a better understanding of the FPN relationships that facilitate the verification with an automatic proof checker.

In a floating-point pen-and-paper proof, it is difficult to be absolutely sure that no special case is forgotten, no inequality is erroneous, and no implicit hypothesis is assumed, etc. All the proofs presented in this paper are verified using a specification of generic floating-point arithmetic [12] and Coq proof assistant [13]. This approach has already been proved successful in hardware or software applications [11], [14], [15], [16]. The drawback is a long and tiresome argumentation versus the proof checker that will ascertain each step of the proof. The corresponding scripts of proofs are available online with the theorems at http://www.netlib.org/fp/fp2.tgz. We indicate for each theorem its Coq name. The developments presented here are located in the `FArgReduct[2, 3, 4].v` files.

The rest of this paper is organized as follows: Section 2 recalls theorems on the number of canceled bits of two close FPNs

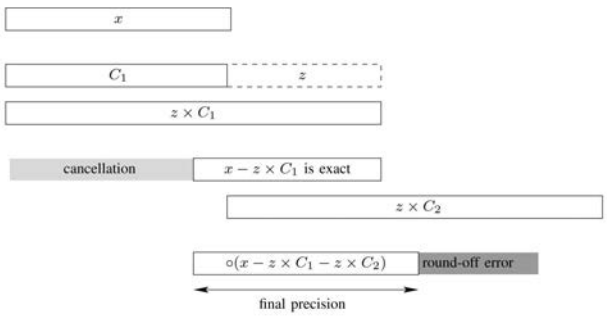

Fig. 3. Argument reduction with exact cancellation in an `fma`.

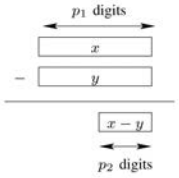

Fig. 4. Extension of Sterbenz's theorem.

(extensions of Sterbenz's theorem [17]). In Section 3, we present the Coq-verified theorem about the correctness of the algorithm that produces z in (1.1) and that satisfies the conditions of the following theorems. The demonstration of the main results and the subsequent choices for R and C_1 and their applications to $C = \ln 2$ for the exponential function and $C = \pi$ for the trigonometric functions are then described in Section 4. Section 5 gives new algorithms and results about the second step of the reduction and choices for C_2. Section 6 concludes the work of this paper.

Notation. Throughout, $\ominus$ denotes the floating-point subtraction. $\{\mathcal{X}\}_{\text{fma}}$ denotes the result by an instruction of the `fma` type, i.e., the exact $\pm a \pm b \times c$ after only one rounding. FPNs use p digits, hidden bit (if any) counted, in the significand or are otherwise explicitly stated. We denote by $\circ(a)$ the FPN obtained from rounding a in the round-to-nearest mode with p digits, and we denote it by $\circ_m(a)$ if we round to m digits instead of p. We denote by $\text{ulp}(\cdot)$ the unit in the last place of a p-digit FPN, and $\text{ulp}^{\circ 2}(\cdot) = \text{ulp}(\text{ulp}(\cdot))$. The smallest (subnormal) positive FPN is denoted by λ.

2 Exact Subtraction Theorems

These theorems will be used in Section 4 to guarantee that there will be enough cancellation in $x - zC_1$ so that it can be computed exactly by one `fma`-type instruction or, equivalently, to assure that $x - zC_1$ fits into one FPN.

A well-known property [17], [18] of the floating-point subtraction is the following:

Theorem 1 (`Sterbenz` *in* `Fprop.v`). *Let x and y be FPNs. If $y/2 \le x \le 2y$, then $x - y$ is a FPN. This is valid with any integer radix $\beta \ge 2$ and any precision $p \ge 2$.*

We extend Sterbenz's theorem to fit the use of an `fma` that may create a higher precision virtual number whose leading digits are canceled to the working precision, as explained in Fig. 4: when x and y are sufficiently near one another, cancellation makes the result exactly fit a smaller precision.

Theorem 2 (`SterbenzApprox2`). *Let x and y be p_1-digit FPNs. If*

$$\frac{y}{1+\beta^{p_2-p_1}} \le x \le (1+\beta^{p_2-p_1})\, y,$$

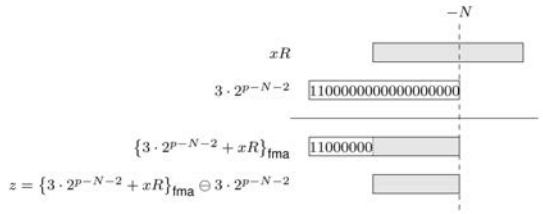

Fig. 5. Algorithm for computing z.

then $x - y$ is a p_2-digit FPN. This is valid with any different significand sizes $p_1, p_2 \geq 2$, and any integer radix $\beta \geq 2$.

The proofs are omitted as they appeared in other publications [8], [19]. It is worth mentioning that Theorem 2 does not require $p_1 \geq p_2$ or $p_2 \geq p_1$.

From now on, all FPNs are binary. The underlying machine hardware conforms to IEEE-754 floating-point standards [20], [21]. This implies that rounding does not introduce a rounding error when the exact result is an FPN. Unless explicitly stated, the default rounding mode is *round to nearest* with ties broken to the even significand.

3 About the Algorithm for z

The computation of z can be done efficiently as

$$z = \{xR + \sigma\}_{\text{fma}} - \sigma, \tag{3.1}$$

where σ is a prechosen constant. The technique is adapted from [1, Chap. 10], which used an idea attributed by the author to C. Roothaan in his work for HP's vector math library for Itanium. The explanation is in Fig. 5. Here, we choose $\sigma = 3 \cdot 2^{p-N-2}$ for a z having its last bit at exponent $-N$.

In realizing (1.1), the intended results are that $z2^N$ is an integer and that $|xR - z| \leq 2^{-N-1}$. We may later use that the precision needed for z is smaller or equal to $p - 2$. Here is the theorem, verified by Coq.

Theorem 3 (`arg_reduct_exists_k_zH`). *Assume*

- $p > 3$,
- *x is a p-bit FPN,*
- *R is a positive normal p-bit FPN,*
- $z = \{3 \cdot 2^{p-N-2} + xR\}_{\text{fma}} \ominus 3 \cdot 2^{p-N-2}$,
- $|z| \geq 2^{1-N}$,
- $|xR| \leq 2^{p-N-2} - 2^{-N}$, *and*
- 2^{-N} *is a FPN.*

Then there exists an integer ℓ satisfying $2 \leq \ell \leq p - 2$ such that

- *$|z2^N|$ is an ℓ-bit integer greater than $2^{\ell-1}$, and*
- $|xR - z| \leq 2^{-N-1}$.

In short, if z is computed as explained and x is not too big, then z is a correct answer, meaning that it fulfills all the requirements that will be needed in Theorem 4 in the next section.

For Intel's double extended precision, this technique is perfectly adapted for range reduction with arguments between -2^{63} and 2^{63} when R is close to the unity. This argument range coincides with what is in Intel's manual [22] for `FSIN`, `FCOS`, `FPTAN`, and `FSINCOS`. A quick justification is that for $C = 2\pi$ and a modest N, say, $N = 0$, $|xR| \lesssim 2^{63}/(2\pi)$ gives $|xR| < 2^{62} - 1$.

For the exponential function, any argument larger than 11,356 overflows in the double extended and quad precisions, and $\ell \leq p - 2$ is easily satisfied.

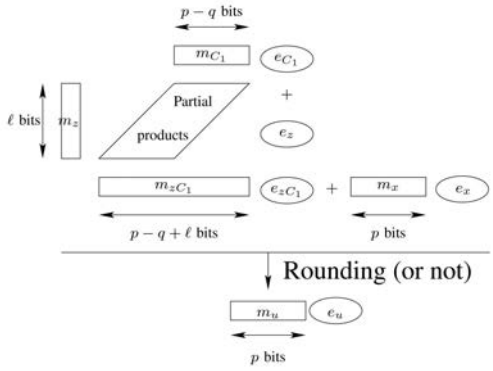

Fig. 6. `fma` used to create and cancel a higher precision virtual number.

4 Main Results

We now present the conditions under which $x - zC_1$ can be represented exactly by an FPN, and thus, it can be computed by $\{x - zC_1\}_{\text{fma}}$ without error. As in Section 1, $R \approx 1/C$, and $C_1 \approx C$. We suppose that $C > 0$ and $C \neq 2^j$ for any j.

The idea is the use of an `fma` that may create a higher precision virtual number that cancels to the working precision. Fig. 6 explains the idea: if z is an ℓ-bit integer and the significand of C_1 uses $p - q$ bits, it takes up to $p - q + \ell$ bits to store the significand of zC_1. And as zC_1 and x are near enough, the final result fits into p bits. The notation m_X stands for the significand of X, and e_X stands for its exponent.

We want to give enough hypotheses on the inputs to guarantee that $x - zC_1$ will be computed without error.

We define the exponent e_R of R as the only integer such that $2^{e_R} < R < 2^{e_R+1}$. We want to set the q least significant bits of C_1 to zero. Since C_1 should be as accurate as possible, we set $C_1 \approx 1/R$ to the nearest FPN with $p - q$ significant bits. From this, we deduce that $2^{-e_R-1} \leq C_1 \leq 2^{-e_R}$ and that the distance between C_1 and $1/R$ is less than half a ulp (in $p - q$ precision); therefore

$$\left|\frac{1}{R} - C_1\right| \leq 2^{-e_R-1-(p-q)}.$$

We now define $\delta = RC_1 - 1$, and we deduce a bound on its magnitude from the previous inequalities:

$$|\delta| \leq 2^{q-p}.$$

Let z be as defined by (1.1) with the conditions on z and s given there. We assume for the moment that $z \neq 0$. Theorem 2 can be used if we bound $x/(zC_1)$ and its reciprocal by $1 + 2^{q-\ell}$. We have the following equalities:

$$\begin{aligned}\frac{x}{zC_1} &= \frac{xR}{zRC_1} \\ &= \frac{z+s}{zRC_1} \\ &= \left(1 + \frac{s}{z}\right)\frac{1}{1+\delta}.\end{aligned}$$

We recall that $z = k2^{-N}$ and that k is an integer using ℓ bits, and we deduce, on the other hand, that

$$2^{-N+\ell-1} \leq |z| < 2^{-N+\ell}$$

to bound

$$\left|\frac{s}{z}\right| \leq 2^{-\ell}.$$

Rewriting the condition in Theorem 2 and taking advantage of the preceding results, we arrive at the point to prove both of the following:

$$\frac{1+2^{-\ell}}{1+|\delta|} \le 1+2^{q-\ell}, \tag{4.1}$$

$$\frac{1+|\delta|}{1-2^{-\ell}} \le 1+2^{q-\ell}. \tag{4.2}$$

Conditions (4.1) and (4.2) are checked using functional analysis on polynomials and linear fractional transformation for any permitted value of $A = 2^{-\ell}$. Since z is both a machine number and a nonzero ℓ-bit FPN, we have $1 \le \ell \le p$. From Section 3, the algorithm used to produce z implies that $\ell \le p-2$. We will use a more generic condition:

$$2^{1-p} \le A = 2^{-\ell} \le \frac{1}{2}.$$

We will now explain what are the successive requirements to guarantee that both (4.1) and (4.2) are fulfilled:

1. *Condition (4.1)*. We want to guarantee that $\frac{1+2^{-\ell}}{1+2^{q-\ell}} \le 1+|\delta|$. The linear fractional transformation

   $$\frac{1+2^{-\ell}}{1+2^{q-\ell}} = \frac{1+A}{1+A2^q}$$

   that we intend to bound is maximized at $A = 2^{1-p}$, and it is sufficient to check if $(1+2^{1-p})/(1+2^{1-p}2^q) \le 1+|\delta|$. We use the bound on $|\delta|$, and we introduce $B = 2^q$. We are left to prove that

   $$(1+2^{1-p})/(1+2^{1-p}B) \le 1 - B2^{-p}.$$

   This is equivalent to checking if the second-order polynomial $2^{1-p}B^2 - B + 2 \le 0$. The inequality is satisfied for B between the two roots $2^{p-2}(1 \pm \sqrt{1-2^{4-p}})$. Thus, it is sufficient to have $B \ge 4$ for all precisions.
2. *Condition (4.2)*. We want to guarantee that $1+|\delta| \le (1+2^{q-\ell})(1-2^{-\ell})$. We introduce A and B as before, so we are left to prove that

   $$1+|\delta| \le (1+AB)(1-A).$$

   We assume that $B \ge 4$ from the preceding paragraph. The polynomial

   $$(1+AB)(1-A) = (1+2^{q-\ell})(1-2^{-\ell})$$

   is minimized at $A = 2^{1-p}$, and it is sufficient to check that $(1+|\delta|) \le (1+2^{1-p}B)(1-2^{1-p})$. From the bound on $|\delta|$, we now have to check that

   $$(1+B2^{-p}) \le (1+2^{1-p}B)(1-2^{1-p}),$$

   which is true for any precision.

This proof is rather long and intricate. We therefore verified it in Coq to be sure that it contains no mistake. Such additional work also gives us more precise and sharp hypotheses than pen-and-paper proofs. All hypotheses have to be clearly written so that the proof can be checked. There is no easy way to say "we assume that there is no Underflow" or "the precision is big enough." This leads to long theorems (at least longer than what we were used to) but precise and correct ones:

Theorem 4 (`Fmac_arg_reduct_correct1`). *Assume*

- $p > 3$,
- x *is a* p*-bit FPN,*
- z *is a FPN,*
- R *is a positive normal* p*-bit FPN,*
- $2 \le q < p-1$,
- C_1 *is the* $(p-q)$*-bit FPN obtained by rounding* $1/R$ *to* $p-q$ *bits using round-to-nearest mode,*
- C_1 *is not exactly a power of 2,*
- $C_1 \ge 2^{p-q+\max(1,N-1)}\lambda$,
- $2 \le \ell \le p-1$,
- $|z2^N|$ *is an* ℓ*-bit integer greater than* $2^{\ell-1}$,
- $|xR - z| \le 2^{-N-1}$, *and*
- $q \le \ell$.

Then, $x - zC_1$ *is a* p*-bit FPN.*

In short, if C_1 is rounded to the nearest from $1/R$ with $p-q$ bits and $q \ge 2$ and z is not too small, then the `fma` does not introduce any round-off error.

Automatic proof checking also prompted us that the exact behavior may be difficult to obtain for $z = 2^{-N}$ and x close to $2^{-N-1}R$. This case was excluded in Theorem 4 under the hypothesis that $2 \le \ell$, but it will be included in the next theorem, which focuses on $q = 2$ as this situation leads to C_1 as close as possible from C, and thus, it has some higher practical value. For completeness and theoretical interest, a theorem similar to Theorem 4 but valid for all $2 \le q \le p-1$ is presented in the Appendix.

Assume that $q = 2$ in the rest of this section. When $z = 2^{-N}$, then $x \le 2C_1 \times 2^{-N}$ as xR is approximated by $z = 2^{-N}$. We can also deduce that

$$\frac{C_1 \times 2^{-N}}{1+2^{2-p}} \le x.$$

When $C_1 \times 2^{-N}/2 \le x$, Sterbenz's theorem (Theorem 1) can be applied, and $x - C_1 \times 2^{-N}$ is an FPN. If not, then

$$\frac{C_1 \times 2^{-N}}{1+2^{2-p}} \le x < \frac{C_1 \times 2^{-N}}{2}.$$

Since C_1 is a $(p-2)$-bit FPN and not exactly a power of 2 as a p-bit FPN, then C_1 is at least 4 ulp's away from a power of 2. This is because as a p-bit FPN, C_1 is worth $2^e \times 1.\mathtt{bb}\cdots\mathtt{b}00$, where at least one of the $\mathtt{b}$'s must be 1; therefore, the C_1 that comes closest to a power of 2 is either $2^e \times 1.0\cdots0100$ or $2^e \times 1.11\cdots100$. Both are 4 ulp's away from a power of 2. This distance and the preceding inequality are enough to guarantee that the exponent of x is the exponent of C_1 minus $N+1$. After a few computations, we finish with $x - C_1 \times 2^{-N}$ being an FPN, regardless of x.

A few peculiar cases have been omitted in the sketch of this proof. Automatic proof checking allows us to trustfully guarantee that these cases have been all checked in our publicly available proof scripts. The only surprising condition is presented in this section. The other cases are easily generalized from Theorems 3 and 4. So just by wrapping these two results together, we can state the following theorem in its full length, verified with Coq.

Theorem 5 (`Fmac_arg_reduct_correct3`). *Assume*

- $p > 3$,
- x *is a* p*-bit FPN,*
- R *is a positive normal* p*-bit FPN,*
- C_1 *is the* $(p-2)$*-bit FPN obtained by rounding* $1/R$ *to* $p-2$ *bits using round-to-nearest mode,*
- C_1 *is not exactly a power of 2,*
- $C_1 \ge 2^{p+\max(-1,N)}\lambda$,
- $z = \left\{3 \cdot 2^{p-N-2} + xR\right\}_{\text{fma}} \ominus 3 \cdot 2^{p-N-2}$,

TABLE 1
Example of Value for $R = \circ(1/C)$, C_1 Rounded to $p-2$ Bits, C_2 Obtained from Algorithm 5.2, and C_3, for $C = \pi$, Easily Leading to $C = 2\pi$ or $C = \pi/2$

Precision	Single	Double	Double extended	Quad
R	$10680707 \cdot 2^{-25}$	$5734161139222659 \cdot 2^{-54}$	$11743562013128004906 \cdot 2^{-65}$	$661103768829069934368299728213873 0 \cdot 2^{-114}$
C_1	$13176796 \cdot 2^{-22}$	$7074237752028440 \cdot 2^{-51}$	$14488038916154245684 \cdot 2^{-62}$	$81560408330151882008337430813741 36 \cdot 2^{-111}$
C_2	$-11464520 \cdot 2^{-45}$	$4967757600021504 \cdot 2^{-105}$	$14179128828124470480 \cdot 2^{-126}$	$93516615446317514493723239679207 68 \cdot 2^{-226}$
C_3	$-15186280 \cdot 2^{-67}$	$7744522442262976 \cdot 2^{-155}$	$10700877088903390780 \cdot 2^{-189}$	$-91863782037025581494013088907961 40 \cdot 2^{-334}$

TABLE 2
Example of Value for $R = \circ(1/C)$, C_1 Rounded to $p-2$ Bits, C_2 Obtained from Algorithm 5.2, and C_3, for $C = \ln(2)$

Precision	Single	Double	Double extended	Quad
R	$12102203 \cdot 2^{-23}$	$6497320848556798 \cdot 2^{-52}$	$13306513097844322492 \cdot 2^{-63}$	$74909009286315393943232627301955 14 \cdot 2^{-112}$
C_1	$11629080 \cdot 2^{-24}$	$6243314768165360 \cdot 2^{-53}$	$12786308645202655660 \cdot 2^{-64}$	$71980518562473539470808149036912 40 \cdot 2^{-113}$
C_2	$-8577792 \cdot 2^{-52}$	$-7125764960002032 \cdot 2^{-106}$	$-15596301547560248640 \cdot 2^{-130}$	$-53812359250046375530745201292023 40 \cdot 2^{-224}$
C_3	$-8803384 \cdot 2^{-72}$	$-7338834209110452 \cdot 2^{-161}$	$-13766585803531045332 \cdot 2^{-192}$	$-94379828466771422085523396350877 88 \cdot 2^{-338}$

- $|xR| \leq 2^{p-N-2} - 2^{-N}$, *and*
- 2^{-N} *is a FPN.*

Then, $x - zC_1$ *is a p-bit FPN.*

In short, if C_1 is rounded to the nearest from $1/R$ with $p-2$ bits and z is computed as usual, then the `fma` does not make any round-off error. In Tables 1 and 2, we present constants R and C_1 for π and $\ln(2)$. These constants are for the exponential and the fast reduction phase of the trigonometric functions [1], [3], [9], [23].

The hypotheses may seem numerous and restrictive, but they are not. As R and C_1 are precomputed, the corresponding requirements can be checked beforehand. Moreover, those requirements are weak: for example, with $0 \leq N \leq 10$ in double precision, we need $C_1 \geq 2^{-1011} \approx 4.5 \cdot 10^{-305}$. There is no known elementary function for which C_1 ever comes near a power of 2. The only nontrivial requirement left is the bound on $|xR|$.

5 Getting More Accurate Reduced Arguments

As we pointed out in the introduction in Section 1, sometimes, the reduced argument $u = x - zC_1$ is not accurate enough due to the limited precision in C_1 as an approximation to C. When this happens, another FPN C_2 containing the lower bits of the constant C has to be made available, and the new reduced argument is now $x - zC_1 - zC_2$. The inaccuracy may also be due to the multiplication by the reciprocal approximation of C, even when C is exact, but this is out of the scope of this paper.

Assume that the conditions of Theorem 5 hold. In particular, C_1 has $p-2$ bits in its significand.

The number $x - zC_1 - zC_2$ can be computed exactly [24] as the sum of two FPNs. But since we know here some conditions on FPNs z, C_1, and C_2, we can obtain the results faster. We propose Algorithm 5.1, inspired from [24], to accomplish such a task. It is built upon two known algorithms:

- $\mathrm{Fast2Mult}(x, y)$, which computes the rounded product of x and y and its error (2 flops) [25] and
- $\mathrm{Fast2Sum}(x, y)$, which computes the rounded sum of x and y and its error (3 flops), under the hypothesis that $x = 0$, $y = 0$, $|x| \geq |y|$, or there exist some integers n_x, e_x, n_y, and e_y such that $x = n_x 2^{e_x}$, $y = n_y 2^{e_y}$, and $e_x \geq e_y$ [12].

Algorithm 5.1 (superaccurate argument reduction): The correctness of this algorithm is only guaranteed under the conditions of Theorem 6. It does not work with any C_1 and C_2!

$$\begin{aligned} u &= \circ(x - zC_1), \\ v_1 &= \circ(u - zC_2), \\ (p_1, p_2) &= \mathrm{Fast2Mult}(z, C_2), \\ (t_1, t_2) &= \mathrm{Fast2Sum}(u, -p_1), \\ v_2 &= \circ(\circ(\circ(t_1 - v_1) + t_2) - p_2). \end{aligned}$$

Theorem 6 (`FArgReduct4.v` file). *Assume*

- $p > 4$,
- x *is a p-bit FPN,*
- R *is a positive normal p-bit FPN,*
- C_1 *is the* $(p-2)$*-bit FPN obtained by rounding* $1/R$ *to* $p-2$ *bits using round-to-nearest mode,*
- C_1 *it is not exactly a power of 2,*
- $z = \left\{3 \cdot 2^{p-N-2} + xR\right\}_{\mathrm{fma}} \ominus 3 \cdot 2^{p-N-2}$,
- $|xR| \leq 2^{p-N-2} - 2^{-N}$,
- 2^{-N} *is a normal p-bit FPN,*
- $C_1 \geq 2^{p+\max(-1,p+N-2)}\lambda$,
- C_2 *is a FPN and an integer multiple of* $8ulp^{\circ 2}(C_1)$,
- $|C_2| \leq 4ulp(C_1)$, *and*
- v_1 *and* v_2 *are computed using Algorithm 5.1.*

Then, Fast2Sum works correctly and we have the mathematical equality $v_1 + v_2 = x - zC_1 - zC_2$ *(all the computations of the last line did indeed introduce no rounding error).*

The first requirements are very similar to the previous ones. The "no-underflow" bound on C_1 has been raised, but it is still easily achieved by real constants. For a typical N between 0 and 10 used by the existing elementary math libraries in IEEE double precision, it suffices that $C \geq 10^{-288}$.

The most important add-ons are the requirements on C_2: it must be much smaller than C_1 (it is near the difference between the constant C and C_1). And C_2 must not be "too precise." In fact, $C_1 + C_2$ will have $2p-4$ bits, as shown in Fig. 7. If by chance, there are a lot of zeros just after C_1, we cannot take advantage of that to get some more precise C_2. This is a real drawback, but it does not happen very often that many zeros are just at these positions.

This algorithm may seem simple, but it is a very powerful tool. It is *exact* and it is *fast*: the generic algorithm [24] costs 20 flops

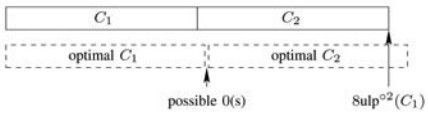

Fig. 7. Respective layouts of our C_1 and C_2 compared to optimal values.

while this one costs only 9 flops! Furthermore, the correcting term can be stored in one FPN instead of two for the generic algorithm.

We can use several constants for C_2 as the conditions on C_2 are rather mild. Some external considerations may call for the largest or the smallest constant allowed for C_2. Algorithm 5.2 gives one way to choose a convenient C_2.

The proof of Theorem 6 is based on a careful study of the possible exponents for the FPNs involved. We first prove that x is an integer multiple of $2^{-N}\mathrm{ulp}(C_1)$. This is done whether z is 2^{-N} or not. We conclude that we correctly use a Fast2Sum operation.

We notice that t_1 and v_1 are integer multiples of $2^{-N-1}\mathrm{ulp}^{\circ 2}(C_1)$ and that $|t_1 - v_1| < 2^{p-N-1}\mathrm{ulp}^{\circ 2}(C_1)$. We deduce that $t_1 - v_1$ fits in an FPN.

The next step is about $t_1 - v_1 + t_2 = u - p_1 - v_1$ being an FPN. We do it similarly. All these quantities are also integer multiples of $2^{-N-1}\mathrm{ulp}^{\circ 2}(C_1)$, and we easily check that $|t_1 - v_1 + t_2| < 2^{p-N-1}\mathrm{ulp}^{\circ 2}(C_1)$.

We finally prove that $t_1 - v_1 + t_2 - p_2 = u - zC_2 - v_1$ fits in an FPN. Its least significant nonzero bit is at most shifted N times down compared to the least significant nonzero bit of C_2. For this reason, we require that C_2 is an integer multiple of $8\mathrm{ulp}^{\circ 2}(C_1)$.

This proof needs a careful study of the relationships between the various FPNs and their exponent values. The formal proof and its genericity allowed us a better understanding of the respective layouts of the FPNs, which is the key of the correctness of Algorithm 5.1.

Algorithm 5.2 (computation of C_2): Let C be the exact constant (for example, π or $\ln 2$).

$$\begin{aligned} R &= \circ_p(1/C), \\ C_1 &= \circ_{p-2} - (1/R), \end{aligned}$$

and take C_2 to be the first many significand bits of $C - C_1$ so that its least nonzero bit must be greater than or equal to $\log_2(\mathrm{ulp}(C_1)) - p + 4 = \log_2(8\mathrm{ulp}^{\circ 2}(C_1))$, e.g.,

$$C_2 = \left\lceil \frac{(C - C_1)}{8\mathrm{ulp}^{\circ 2}(C_1)} \right\rfloor 8\mathrm{ulp}^{\circ 2}(C_1),$$

where $\lceil \cdot \rfloor$ is one of the round-to-integer operations.

The constant computed has all the expected properties, except that we do not know for sure that $|C_2| \leq 4\mathrm{ulp}(C_1)$. Note that C_1 is not given simply by rounding C, but rather, $C_1 = \circ(1/\circ(\frac{1}{C}))$.

Theorem 7 (`gamma2_1e`). *Assume*

- $p > 3$,
- *C is a real positive constant,*
- *R is the p-bit FPN obtained by rounding $1/C$ to p bits using round-to-nearest mode,*
- *R is a positive normal p-bit FPN,*
- *C_1 is the $(p-2)$-bit FPN obtained by rounding $1/R$ to $p-2$ bits using round-to-nearest mode,*
- *C_1 is not exactly a power of 2, and*
- $C_1 \geq 2^{p-1}\lambda$.

Then, $|C - C_1| \leq 4ulp(C_1)$.

As C is not too far from C_1, we have that $C \leq 2^{p+1}\mathrm{ulp}(C_1)$. We now bound $C - C_1$ to finish all proofs:

$$\begin{aligned} |C - C_1| &\leq \left|C - \frac{1}{R}\right| + \left|\frac{1}{R} - C_1\right| \\ &\leq \frac{C}{R}\left|R - \frac{1}{C}\right| + \left|\frac{1}{R} - C_1\right| \\ &\leq \frac{C}{R}\mathrm{ulp}(R)/2 + 4\mathrm{ulp}(C_1)/2 \\ &\leq C2^{-p-1} + 2\mathrm{ulp}(C_1). \end{aligned}$$

This means that the formula for C_2 given above yields an FPN fulfilling the requirements of Theorem 6.

6 Conclusions

We have presented Coq-verified theorems that prove the correctness and effectiveness of a much faster technique based on the commonly used argument reduction in elementary function computations on machines that have hardware support for `fma` instructions. The conditions of these theorems are easily met as our analysis indicates. While we have showed that it is not always possible to use the most accurate parameters under all circumstances, an almost-best possible selection can be used at all times: to zero out the last 2 bits.

We have presented also a very accurate second-step argument reduction. We provide a way to compute C_2, which is not the most precisely possible but is usually 2 bits away from it (and can be rounded as needed by the programmer). The most interesting part is the possibility to compute with FPNs the exact error of the second step of the argument reduction and the fact that this error is exactly representable by only one FPN. It makes the third step unexpectedly easy as we have a mathematical equality between the computed FPNs and a very good approximation of $x - zC$ (with a known error).

Except for the computation of C_2, all the rounding used should be rounding to nearest, ties to even. But our proofs are generic enough to show that our results still hold when using rounding to nearest, where cases of ties can be decided in any coherent way [26]. This includes rounding to nearest, ties away from zero that is found in the revision of the IEEE-754 standard.

The formal verification forces us to provide many tedious details in the proofs but gives us a guarantee on our results. The proposed theorems are sufficient in the sense that effective parameters for efficient argument reductions can be obtained without any difficulty.

Our theorems provide us with sufficient conditions for $x - zC_1$ to be an FPN. This means that $x - zC_1$ could be an FPN even when one or more of the conditions fails for some specific values of C, C_1 and R, as published in the past [1], [7]. We may work on this in the future even though there is only a limited space for improvement as only the last two bits of C_1 can be changed to make the constant more accurate.

The algorithms proved can be applied to any floating-point format (IEEE single, double, or extended, for example). Intuitively, the correctness of these algorithms should come as natural. Nevertheless, rigorous proofs are not trivial due to a few special cases that could have been easily dismissed by hand-waving proofs.

Appendix

Theorem 4 can be used for any value of $2 \leq q \leq p-1$. In most case, users are interested in the smallest possible value of q because that will give a more accurate C_1 and, consequently, a more accurate

reduced argument. For this reason, we proved Theorem 5 for $q = 2$. The following theorem is under the hypothesis that

$$RC_1 \leq 1,$$

while $2 \leq q \leq p-1$ still. This add-on is enough to guarantee cases that are left over by Theorem 4.

Theorem 8 (`Fmac_arg_reduct_correct2`). *Assume*

- $p > 3$,
- $2 \leq q < p-1$,
- x *is* p*-bit FPN*,
- R *is a positive normal* p*-bit FPN*,
- C_1 *is the* $(p-q)$*-bit FPN obtained by rounding* $1/R$ *to* $p-q$ *bits using round-to-nearest mode*,
- C_1 *is not exactly a power of 2*,
- $C_1 \geq 2^{p-q+\max(1,N-1)}\lambda$,
- $z = \left\{3 \cdot 2^{p-N-2} + xR\right\}_{\text{fma}} \ominus 3 \cdot 2^{p-N-2}$,
- 2^{-N} *is a FPN*,
- $|xR| \leq 2^{p-N-2} - 2^{-N}$,
- $RC_1 \leq 1$.

Then, $x - zC_1$ *is a* p*-bit FPN.*

We essentially need to consider how to make R and C_1 satisfy this new constraint. Since there is no strict connection between R and C_1 on one hand and between R and C on the other hand, we can either use R to be the correctly rounded FPN nearest $1/C$ or, alternatively, add or subtract one or a few ulp's so that the additional inequality is met.

ACKNOWLEDGMENTS

This work was supported in part by the US National Science Foundation under Grants DMS-0510664 and DMS-0702335 and by the Region Languedoc Roussillon of France. This work was also supported in part by the PICS 2533 of the CNRS and the project EVA-Flo of the ANR.

REFERENCES

[1] P. Markstein, *IA-64 and Elementary Functions: Speed and Precision.* Prentice Hall, 2000.

[2] N. Brisebarre, D. Defour, P. Kornerup, J.-M. Muller, and N. Revol, "A New Range-Reduction Algorithm," *IEEE Trans. Computers*, vol. 54, no. 3, pp. 331-339, Mar. 2005.

[3] J.-M. Muller, *Elementary Functions, Algorithms and Implementation.* Birkhauser, http://www.springer.com/west/home/birkhauser/computer+science?SGWID=4-40353-22-72377986-0, 2006.

[4] R.-C. Li, "Near Optimality of Chebyshev Interpolation for Elementary Function Computations," *IEEE Trans. Computers*, vol. 53, no. 6, pp. 678-687, June 2004.

[5] W.J. Cody and W. Waite, *Software Manual for Elementary Functions.* Prentice Hall, 1980.

[6] P.W. Markstein, "Computation of Elementary Functions on the IBM RISC System/6000 Processor," *IBM J. Research and Development*, vol. 34, no. 1, pp. 111-119, http://www.research.ibm.com/journal/rd/341/ibmrd3401N.pdf, 1990.

[7] S. Story and P.T.P. Tang, "New Algorithms for Improved Transcendental Function on IA-64," *Proc. 14th IEEE Symp. Computer Arithmetic (ARITH '99)*, I. Koren and P. Kornerup, eds., pp. 4-11, http://computer.org/proceedings/arith/0116/0116toc.htm, 1999.

[8] R.-C. Li, S. Boldo, and M. Daumas, "Theorems on Efficient Argument Reductions," *Proc. 16th IEEE Symp. Computer Arithmetic (ARITH '03)*, J.-C. Bajard and M. Schulte, eds., pp. 129-136, http://hal.archives-ouvertes.fr/hal-00156244, 2003.

[9] W. Kahan, *Minimizing* $q \times m - n$, http://www.cs.berkeley.edu/~wkahan/testpi/nearpi.c, 1983.

[10] J. Harrison, "A Machine-Checked Theory of Floating Point Arithmetic," *Proc. 12th Int'l Conf. Theorem Proving in Higher Order Logics (TPHOLs '99)*, Y. Bertot, G. Dowek, A. Hirschowitz, C. Paulin, and L. Théry, eds., pp. 113-130, http://www.cl.cam.ac.uk/users/jrh/papers/fparith.ps.gz, 1999.

[11] J. Harrison, "Formal Verification of Floating Point Trigonometric Functions," *Proc. Third Int'l Conf. Formal Methods in Computer-Aided Design (FMCAD '00)*, W.A. Hunt and S.D. Johnson, eds., pp. 217-233, 2000.

[12] M. Daumas, L. Rideau, and L. Théry, "A Generic Library of Floating-Point Numbers and Its Application to Exact Computing," *Proc. 14th Int'l Conf. Theorem Proving in Higher Order Logics (TPHOLs '01)*, pp. 169-184, http://hal.archives-ouvertes.fr/hal-00157285, 2001.

[13] Y. Bertot and P. Castéran, "Interactive Theorem Proving and Program Development. Coq'Art: The Calculus of Inductive Constructions," *Texts in Theoretical Computer Science*. Springer, 2004.

[14] V.A. Carreño and P.S. Miner, "Specification of the IEEE-854 Floating-Point Standard in HOL and PVS," *Proc. 1995 Int'l Workshop Higher Order Logic Theorem Proving and Its Applications*, http://shemesh.larc.nasa.gov/fm/ftp/larc/vac/hug95.ps, 1995.

[15] D.M. Russinoff, "A Mechanically Checked Proof of IEEE Compliance of the Floating Point Multiplication, Division and Square Root Algorithms of the AMD-K7 Processor," *LMS J. Computation and Math.*, vol. 1, pp. 148-200, http://www.onr.com/user/russ/david/k7-div-sqrt.ps, 1998.

[16] J. Harrison, "Floating Point Verification in HOL Light: The Exponential Function," Technical Report 428, Univ. of Cambridge Computer Laboratory, http://www.cl.cam.ac.uk/users/jrh/papers/tang.ps.gz, 1997.

[17] P.H. Sterbenz, *Floating Point Computation.* Prentice Hall, 1974.

[18] D. Goldberg, "What Every Computer Scientist Should Know about Floating Point Arithmetic," *ACM Computing Surveys*, vol. 23, no. 1, pp. 5-47, http://doi.acm.org/10.1145/103162.103163, 1991.

[19] S. Boldo and M. Daumas, "Properties of Two's Complement Floating Point Notations," *Int'l J. Software Tools for Technology Transfer*, vol. 5, nos. 2/3, pp. 237-246, http://hal.archives-ouvertes.fr/hal-00157268, 2004.

[20] D. Stevenson et al., "An American National Standard: IEEE Standard for Binary Floating Point Arithmetic," *ACM SIGPLAN Notices*, vol. 22, no. 2, pp. 9-25, 1987.

[21] W.J. Cody, R. Karpinski et al., "A Proposed Radix and Word-Length Independent Standard for Floating Point Arithmetic," *IEEE Micro*, vol. 4, no. 4, pp. 86-100, 1984.

[22] *Pentium Pro Family: Developer's Manual*, Programmer's Reference Manual, Intel, ftp://download.intel.com/design/pro/manuals/24269101.pdf, 1996.

[23] K.C. Ng, *Argument Reduction for Huge Arguments: Good to the Last Bit*, work in progress, http://www.validgh.com/arg.ps, 1992.

[24] S. Boldo and J.-M. Muller, "Some Functions Computable with a Fused-MAC," *Proc. 17th IEEE Symp. Computer Arithmetic (ARITH '05)*, P. Montuschi and E. Schwarz, eds., pp. 52-58, http://csdl.computer.org/comp/proceedings/arith/, 2005.

[25] A.H. Karp and P. Markstein, "High-Precision Division and Square Root," *ACM Trans. Math. Software*, vol. 23, no. 4, pp. 561-589, Dec. 1997.

[26] J.F. Reiser and D.E. Knuth, "Evading the Drift in Floating Point Addition," *Information Processing Letters*, vol. 3, no. 3, pp. 84-87, 1975.